

Luzzu Quality Metric Language – A DSL for Linked Data Quality Assessment

Jeremy Debattista, Christoph Lange, Sören Auer

University of Bonn & Fraunhofer IAIS
 {debattis, lange, auer}@cs.uni-bonn.de

Abstract The steadily growing number of linked open datasets brought about a number of reservations amongst data consumers with regard to the datasets' quality. Quality assessment requires significant effort and consideration, including the definition of data quality metrics and a process to assess datasets based on these definitions. Luzzu is a quality assessment framework for linked data that allows domain-specific metrics to be plugged in. In this paper we describe the Luzzu Quality Metric Language (LQML), a domain specific language (DSL) whose purpose is to enable non-programming domain experts to define quality metrics for the assessment of linked open datasets. LQML offers notations, abstractions and expressive power, focusing on the representation of quality metrics. It provides expressive power for defining sophisticated quality metrics. Its integration with Luzzu enables their efficient processing and execution and thus the comprehensive assessment of extremely large datasets in a streaming way. We also describe a novel ontology that enables the reuse, sharing and querying of such definitions. Finally, we evaluate the proposed DSL against the cognitive dimensions of notation framework.

Keywords: data quality, quality metrics, linked data, domain specific language

1 Introduction

In May 2007 the first version of the Linked Open Data cloud [5] was published. Following Linked Data principles¹, 12 datasets were initially added to the LOD cloud. The cloud saw an increase during the years, and the data provider² count is 570 (as of August 2014). Furthermore, an investigation (from February 2015) at the data catalogue portal datahub.io showed that 1,309 out of 9,262 datasets are tagged with the *lod* or *format-rdf* tags. Moreover, other linked datasets might be available in the Web of Data but are not catalogued, thus they are not easily discovered by data consumers. Schmachtenberg et al. [19] compiled a list of uncatalogued datasets from various W3C mailing list communications and the 2012 Billion Triple Challenge.

However, the increase of linked open datasets brought about a number of reservations amongst data consumers with regard to the datasets' quality. Hitzler and Janowicz [11] state that linked open datasets have a reputation of being of poor quality. Consequently, if we manage to systematically assess and improve data quality of LOD,

¹ <http://www.w3.org/DesignIssues/LinkedData.html>

² Each data provider, identified by its pay-level domain (e.g. <http://dbpedia.org>), can have more than one dataset; 1014 datasets were discovered in 2014

many Linked Open Data applications can be better supported and contribute to establishing the 4th Big Data aspect – Veracity.

Indubitably, quality assessment requires a lot of effort and consideration before processing a dataset. Quality is commonly described as *fitness for use* [14]. Although domain experts can decide on a number of quality factors of a particular dataset, in the end it is up to data consumers to see if a dataset is suitable for their use case or not. Providing measures on the quality of a published linked open dataset should be part of the publishing lifecycle. Various research work defines a number of quality factors pertinent to linked open datasets [2,10,12]. Zaveri et al. [20] provide an overall systematic review of such quality metrics. Additionally, domain specific quality metrics are required to have a more comprehensive view of the dataset’s quality. For a cultural heritage dataset, for example, the ratio of resources being linked to an *Integrated Authority File* (e.g. *GND*³) is of crucial importance.

*Luzzu*⁴, is an extensible quality assessment framework for linked open datasets. Linked Data quality metrics can be added to Luzzu by various third parties (such as programmers and data enthusiasts). It often occurs that *data scientists*, whose spectrum ranges from data publishers and consumers to domain experts and knowledge engineers, might not be confident in programming using traditional third generation languages. Nevertheless, they are considered to be the ideal drivers for defining domain specific quality metrics, which can be used on linked open datasets.

The main contribution of this article is the definition and implementation of the *Luzzu Quality Metric Language* (LQML), a domain specific language (DSL) that enables declarative definition of quality metrics for Luzzu (cf. Section 2). LQML offers notations, abstractions and expressive power, focusing on a the representation of quality metrics for Linked Dataset assessment. A particular challenge in the definition of LQML was to balance between providing the expressive power for defining sophisticated quality metrics on the one hand and ensuring their efficient processing and execution on the other hand. As a result, our implementation enables the comprehensive assessment of extremely large datasets with respect to many quality metrics in a streaming way. LQML is designed in a way that metrics can be written by non-programmers that are experts in the domain (cf.[8,13,18]). Hudak [13] suggests that DSLs have the potential to improve productivity in the long run and with LQML we aim to contribute to overcoming one of the main problems of Linked Open Data – data quality.

We also define a new vocabulary to enable the reuse, sharing and querying of LQML metrics in a semantic manner (cf. Section 3). The usability of the LQML is systematically assessed (cf. Section 4) against the “cognitive dimensions of notation” (CD) evaluation framework. These dimensions provide a comprehensive view of how users can manage and use a defined language. We also briefly outline the state-of-the-art in domain specific languages (cf. Section 5) and concluding remarks and an outlook on future work are discussed in Section 6.

³ German “Gemeinsame Normdatei”; see <http://www.dnb.de/EN/gnd>

⁴ <http://eis-bonn.github.io/Luzzu>

2 Luzzu Quality Metric Language

The *Luzzu Quality Metric Language* (LQML) is a structural declarative language that enables the definition of quality metrics (called *blueprints*) in Luzzu. Based on our experience from the use cases of the DIACHRON FP7 EU project⁵, we anticipate that most domain-specific quality metrics are very similar structure-wise, with minor changes required only in the rules' conditions.

2.1 Analysis

Data quality assessment varies from one domain to another. Although there exist a number of generic quality metrics as defined in [20], different domains might require the assessment of different features. For example, where in geographical datasets the properties `geo:long` and `geo:lat` are absolutely required for resources that are defined as a place (such as country and city), these properties might be redundant in health oriented datasets. The idea of LQML is that data scientists can define various quality metrics over a dataset (or a domain of datasets). These declarative definitions are translated into Java byte-code (see Section 2.3) and integrated within the Luzzu framework. For the proposed domain specific language, we identified a domain terminology based on quality metrics required by pilot partners in the DIACHRON project.

Use case overview

EBI – One of the services of the European Bioinformatics Institute (EBI⁶) is to provide linked datasets to the scientific community, with their main development focusing around the Experimental Factor Ontology (EFO). The EFO ontology is then used to annotate data in databases at the EBI. EFO is an evolving ontology by nature and concepts from external ontologies are constantly being added (or replaced) in the EFO.

Data Publica – This French company⁷ provides a number of data services, which include the management of the largest and most complete directory of electronic data in France. This directory covers all data available in France (private and public), annotating it with relevant metadata, and making it available to the public through various means (search engines, visualisations, etc.).

Domain Terminology: A typical quality metric definition for linked open datasets consists of a *pattern matching condition*, (i.e. matching the subject (`?s`), predicate (`?p`), object (`?o`), or a mixture of these three with possibly advanced inspection), and an *consequent action*. This resembles the traditional *if...then* statements of programming languages. The full representation of an LQML metric definition is termed as *blueprint*. The feature model in Figure 1 describes the features required to create a quality metric blueprint. A blueprint description should have enough information to assess a dataset based on the quality criteria (*Pattern Matching Rules* in Figure 1), and to enable the

⁵ <http://diachron-fp7.eu>

⁶ <http://www.ebi.ac.uk>

⁷ <http://www.data-publica.com>

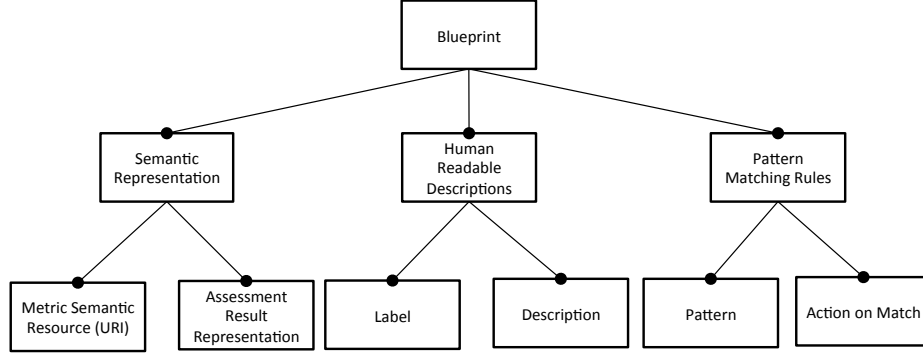


Figure 1. Feature Model for blueprints.

semantic description (*Semantic Representation*) of the quality metadata for the criteria in question. A description should also have a *Human-Readable Description*. This is required since blueprints will be shared amongst different data scientists and thus would enable anyone to understand complex patterns and actions. All features are necessary in order to create one blueprint.

In a more formal manner, let r be the root feature of a *blueprint*. f_1, f_2, f_3 represent the *Semantic Representation* feature, *Human Readable Description* feature, and *Pattern Matching Rules* feature respectively such that:

$$r \longleftrightarrow f_1 \wedge f_2 \wedge f_3 \quad (1)$$

meaning that f_1, f_2, f_3 are mandatory features of r .

Similarly,

$$f_1 \longleftrightarrow f_4 \wedge f_5 \quad (2)$$

where f_4 and f_5 represent the *Metric Semantic Resource (URI)* and *Assessment Result Representation* features respectively;

$$f_2 \longleftrightarrow f_6 \wedge f_7 \quad (3)$$

where f_6 represents the *Label* feature and f_7 the *Description* feature;

$$f_3 \longleftrightarrow f_8 \wedge f_9 \quad (4)$$

where f_8 represents the *Pattern* feature and f_9 the *Action on Match* feature.

2.2 Design

Having identified the features for the proposed domain specific language, we here concisely describe its design and its features. Mernik et al. [18] describe a number of design patterns, three based on *language exploitation* (designs based on existing languages),

and another one for *language invention*. Our proposed DSL is based on the *language invention* design pattern, where we fuse a number of specific terms (such as *typeOf*) from the use cases available together with variable binding expressions used in the syntax of SPARQL (i.e. $?s$ $?p$ $?o$) to refer to specific elements in a triple.

Quality Metric (Blueprint) Structure: A blueprint definition of a metric starts with the `def` keyword and has a rule semantics. Each blueprint consists of the three features mentioned in Section 2.1.

Pattern Matching Rules Feature: Declarative patterns start with the keyword `match` (*Pattern* feature). If a triple matches the given condition, the given `action` (*Action on Match* feature) is triggered. Any input triple (t_s, t_p, t_o) is matched against the conditions that follow the `match` keyword, enclosed into curly brackets $\{ \}$. A rule can have one or more conditions. Conditions can be connected via the *logical and* ($\&$) operator or the *logical or* ($|$) operator. Table 1 shows possible conditions.

Name	Description
	checks the type of subject or object.
<code>typeof(?s ?o)</code>	<code>typeof(?s) == <U></code> translates to the triple pattern “ $?s$ a $<U>$ ”; <code>typeof(?o) == <U></code> translates to the triple pattern “ $?o$ a $<U>$ ”.
<code>?s ?p == <U></code>	matches the subject ($?s$) or the predicate ($?p$) against a given IRI ($<U>$).
<code>?o == <U> x</code>	matches the object ($?o$) against a given IRI ($<U>$) or a literal (x).

Table 1. Pattern Matching Conditions

A condition can trigger one or more of the following actions:

- `map(?s, ?o)` adds the subject and the object to a hash map as key/value (where the value is a list of objects);
- `count` increments a counter;
- `unique(?s | ?p | ?o)` increments a counter only if a unique instance of $?s$, $?p$, or $?o$ is encountered.

In order not to limit LQML expressiveness, programmers can develop custom functions which can then be imported into Luzzu. This enables data scientists to automatically use the imported functions in their defined `match` pattern. Many domain specific languages, including XPath (cf. Section 5), not only have built-in functions⁸, but also enable implementations to provide additional external functions.

Human Readable Descriptions: Descriptive human-readable comments are also required in these blueprints. We provide the keywords `label` and `description` to provide the metric’s name and its textual description; they translate to `rdfs:label` and `rdfs:comment`.

⁸ <http://www.w3.org/TR/xpath-30/#id-function-calls>

Semantic Representation: The definition also expects other information that describes a quality metric. The `metric` (*Metric Semantic Resource (URI)* feature) keyword expects a quality metric resource URI. These resources are defined in a vocabulary that extends the Dataset Quality Ontology (daQ). The `finally` (*Assessment Result Representation* feature) keyword takes a defined function, and returns an output value, which is used as daQ observation value.

The `finally` keyword can have one of the following parameters:

- `actionresult(x)` takes the value of the action, where x stands for `map`, `count`, or `unique`;
- `ratio(x,y)` takes two parameters, which can either be integer or float numbers, or even a function (e.g. `count`) that returns a numeric value. The ratio function divides x by y .

2.3 Implementation

The LQML grammar is implemented in JavaCC (Java Compiler Compiler)⁹. JavaCC is a parser generator and a lexical analyser, where the grammar is specified in EBNF notation. Blueprints defined in LQML are interpreted by the JavaCC compiler where each blueprint is then interpreted and transformed into a Java class during Luzzu's runtime.

The following listing shows the EBNF grammar for the main parts of the LQML syntax.

```
<Definition> ::= <Def> <Metric> <Label> <Description> <Match> <Action>
               <Finally>

<Def> ::= "def" <LBrace> <Strict_Str> <RBrace> <Colon>

<Metric> ::= "metric" <LBrace> <IRIref> <RBrace> <SemiColon>

<Match> ::= "match" <LBrace> (<Condition>)+ <RBrace>

<Condition> ::= <LParen> <TypeOf> | <DefinedFunction> | <other> <RParen>
               (<logical_operator>)*

<TypeOf> ::= "typeof" <LParen> "?s" <RParen> <boolean_operator> <IRIref>

<other> ::= <LParen> "?s" <boolean_operator> <IRIref> <RParen>
           | <LParen> "?p" <boolean_operator> <IRIref> <RParen>
           | <LParen> "?o" <boolean_operator> ( <IRIref> | <Quoted_Str> ) <RParen>

<DefinedFunction> ::= <Strict_Str> "(" ("?s" | "?p" | "?o")* ")"

<IRIref> ::= refer to RFC 3987 [9]

<Action> ::= "action" <LBrace> ((<Map> | <Count> | <Unique>)("," )* )+
               <RBrace>

<Map> ::= "map" <LParen> ("?s" | "?p" | "?o") ("?s" | "?p" | "?o") <RParen>

<Count> ::= "count"

<Unique> ::= "unique" <LParen> ("?s" | "?p" | "?o") <RParen>

<Finally> ::= "finally" <LBrace> (<Number> | <ActionResult> <Ratio>)+
               <RBrace>
```

⁹ <https://java.net/projects/javacc>

```

<ActionResult> ::= "actionresult" <LParen> ("map" | "count" | "unique")
                  <RParen>

<Ratio> ::= "ratio" <LParen> (<Number> | <NumericFunction>) ", " (<Number> |
                  <NumericFunction>) <RParen>

<NumericFunction> ::= <Map> | <Count> | <Unique>

```

Listing 1. LQML EBNF grammar

External Functions: External functions, defined as Java classes, are preloaded into Luzzu beforehand. These can only be used within a match pattern. The structure (as described in the EBNF <DefinedFunction>) requires a function name (as a string) and zero or more variables.

2.4 Blueprint Examples

In order to keep up the quality within the EFO, domain experts from the EBI (cf. Section 2.1) defined relevant quality metrics. One relevant metric is that they identify a percentage of how many resources are actually defined as sub-classes (`rdfs:subClassOf`) of other classes. Listing 2 shows an LQML metric definition for the above.

```

def{SubClassCounter};
metric{<http://www.example.org/ebiqm#SubClassCountingMetric>;
label{"SubClassCountingMetric"};
description{"Provides a measure for counting the number of resources that
are defined as sub-classes"};
match{(?p == <http://www.w3.org/2000/01/rdf-schema#subClassOf>)};
action{count, unique(?s)};
finally{ratio(actionresult(count), actionresult(unique))}.

```

Listing 2. EBI Use Case Example in LQML

One of Data Publica's requirements is that each resource they define has a human readable description or label. This means that they quantify a percentage of how many resources have either an `rdfs:label` or an `rdfs:comment` defined. This metric is defined by LQML in Listing 3.

```

def{HumanReadableLabel};
metric{<http://www.example.org/dpqm#SubClassCountingMetric>;
label{"Human Readable Labelling Metric"};
description{"Provides a measure for identifying the ratio of human readable
labels of defined resources in a dataset?";
match{(typeof(?s) == <http://www.example.org/dp#Class>) && ((?p == <http://
www.w3.org/2000/01/rdf-schema#label>) || (?p == <http://www.w3.org
/2000/01/rdf-schema#comment>))});
action{count, unique(?s)};
finally{ratio(actionresult(count), actionresult(unique))}.

```

Listing 3. Data Publica Use Case Example in LQML

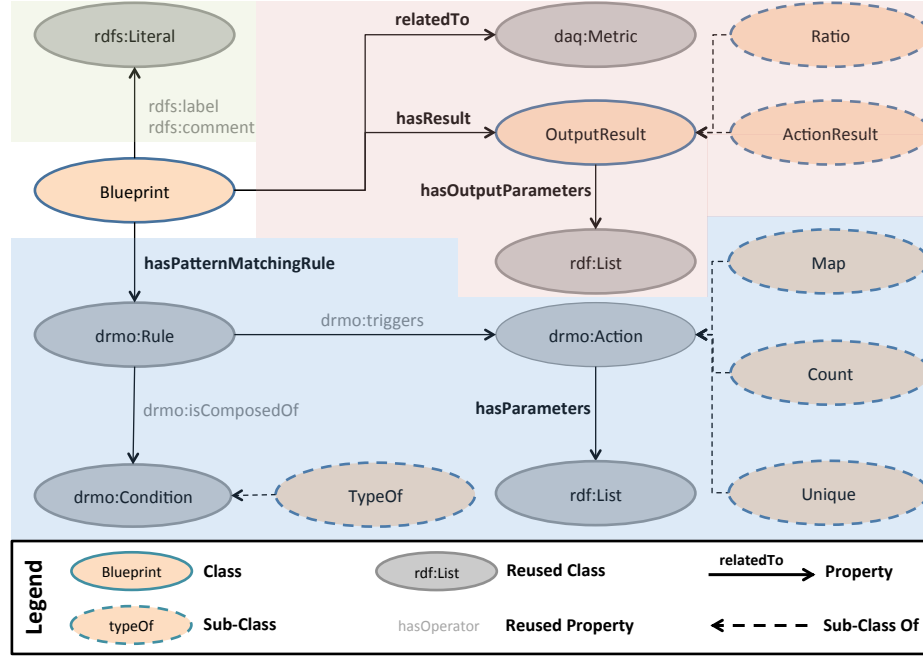


Figure 2. Luzzu Blueprint Ontology.

3 Sharing Blueprints – Luzzu Blueprint Ontology

We envisage that blueprint descriptions can be stored and shared in a common pool of metrics, similar as how different users of the IFTTT service¹⁰ can share rule recipes. Shared blueprints can be either reused or modified to fit the purpose of another use case. The LQML language itself does not enable such sharing to be done with ease. For this purpose, we propose an ontology, the Luzzu Blueprint Ontology (prefix: `lbo`), which facilitates the semantic representation of LQML blueprints. Exploiting this option, the ontology enables us to distribute blueprints as semantic resources. Moreover, these semantic resources are easily queried and visualised.

In line with the Semantic Web principles, the Luzzu Blueprint Ontology (depicted in Figure 2) reuses known concepts and domain specific ontologies.

Listing 4 shows an RDF definition of the *root* feature (cf. Figure 1), `Blueprint` (defined as `lbo:Blueprint` in the proposed ontology).

```
lbo:Blueprint a rdfs:Class .
```

Listing 4. Defining the root feature *r* in Turtle notation

¹⁰ *If This Than That* is an online service allowing users to create simple rules that trigger events: <https://ifttt.com/>

The vocabulary incorporates the three main *features* described in Section 2.1; *Semantic Representation* (highlighted in light red – top right), *Human Readable Descriptions* (highlighted in light green – top left), and *Pattern Matching Rules* (highlighted in light blue – bottom part). Therefore, the formal definitions described in Equations 1, 2, 3, and 4 are given a semantic RDF definition with the Luzzu Blueprint Ontology.

For the *human readable descriptions*, we make use of the standard `rdfs:label` and `rdfs:comment` properties to represent the blueprint’s label and description respectively. From the W3C RDF Schema definition of `rdfs:label` and `rdfs:comment`, these two properties are used to provide a human-readable name and description of a resource respectively. Both properties must have an `rdfs:Resource` as a domain (which instances of `lbo:Blueprint` automatically are) and a literal (`rdfs:Literal`) as its range, i.e. a textual value. These two properties facilitate the Semantic Web notation for Equation 3.

The *semantic representation* (cf. Listing 5) of a quality metric is represented in an instance of the blueprint ontology using the proposed properties `lbo:relatedTo` and `lbo:hasResult`. The former links a blueprint instance to a `daq:Metric` resource. The latter represents the resulting (finally in terms of LQML) output. For this purpose we also introduced two sub-classes of the concept `lbo:OutputResult`; `lbo:Ratio` and `lbo:ActionResults`. The semantics for the latter sub-classes were discussed in Section 2.2. An `lbo:OutputResult` can have one or more output parameters. The proposed property `lbo:hasOutputParameters` expects a resource of type `rdf:List` as its range.

```
# Representing the Metric Semantic Resource (URI) feature f4
lbo:relatedTo a rdf:Property ;
  rdfs:domain lbo:Blueprint ;
  rdfs:range daq:Metric .

# Representing the Assessment Result Representation feature f5
lbo:hasResult a rdf:Property ;
  rdfs:domain lbo:Blueprint ;
  rdfs:range lbo:OutputResult .

# ... definitions of OutputResult, subclasses and hasOutputParameters
property.
```

Listing 5. Defining the *Semantic Representation* feature, Equation 2, using a Semantic Web notation

The Digital.me Rule Management ontology (DRMO) enables users to express rules in terms of resources and concepts that are available in a personal knowledge base [7]. Although it is expected to operate in a closed-world environment, its flexibility of being a domain-independent ontology enables its reuse in other scenarios. The DRMO concepts are inspired by the Event-Condition-Action (ECA) pattern, where the latter pattern is used in event-driven architectures. In light of the *pattern matching rules* feature, we make use of the DRMO concepts, creating a sub-class of `drmo:Condition`, `lbo:TypeOf`; and three sub-classes of `drmo:Action`, namely `lbo:Map`, `lbo:Count`, and `lbo:Unique`. The representation of a condition (the *Pattern* feature – f_8), is left intact. On the other hand, we extended the concept `drmo:Action` the *Action on Match* feature – f_9) with a property named `lbo:hasParameters`, where the

range of the newly proposed property is a resource of type `rdf:List`. The property `hasPatternMatchingRule` (cf. Listing 6) defines the described extended `drmo:Rule` of an `lbo:Blueprint`. The semantics of the `drmo:Rule` concept, together with the properties `drmo:isComposedOf` and `drmo:triggers`, gives a semantic definition for Equation 4.

```
# Representing the relationship between the root feature (r) and the Pattern
  Matching Rules feature f3
lbo:hasPatternMatchingRule a rdf:Property ;
  rdfs:domain lbo:Blueprint ;
  rdfs:range drmo:Rule .

# ... definitions of subclasses for Action (Map, Count, Unique), and
  hasParameters extension to DRMO.
```

Listing 6. Defining the *Pattern Matching Rules* feature f_3

Debattista et al. [7] provide a mechanism that transforms `drmo:Rules` into SPARQL queries. Therefore, blueprint patterns can be transformed into SPARQL queries and re-used even within quality frameworks, such as RDFUnit [16], employing a SPARQL engine as their main assessment tool. This makes Luzzu blueprints interoperable with other quality assessment frameworks.

4 Evaluation

In order to assess the usability of LQML, we gauge the language systematically against the “cognitive dimensions of notation” (CD) evaluation framework, a methodology developed in [3]. This evaluation framework has previously been applied to Semantic Web languages (e.g. [17]). These dimensions provide a comprehensive view of how users can manage and use a defined language. Each dimension describes a specific aspect in relation to the language notation. Blackwell and Green [4] describe the following 13 dimensions:

Viscosity questions the effort required by the user to lead out a change.

Assessment: LQML metrics can be defined using a simple text editor. Each statement is defined for a particular definition (blueprint) and is not related to other definitions. Therefore, changing a statement in a definition does not require a change in any other place, thus resulting in a low viscosity.

Premature Commitment measures any planning required before leading out a task.

Assessment: Based on declarative programming, LQML users only require to define rules based on the patterns they want to match. Also, declarations are not required before a blueprint definition. The only premature commitment is that metrics have to be defined in an ontology (whose URI is defined in the blueprint definition) based on the daQ ontology.

Hidden Dependencies measures if dependencies are specifically indicated in all existing directions.

Assessment: Blueprint definitions cannot be connected to each other, therefore each definition has a fixed rule and action, together with other descriptions.

Error-proneness measures the possibility of users making mistakes while using the language.

Assessment: A definition is made up of only six components. This means the learning curve is not too steep. However, since these six components must always be fixed in the same order, i.e. `metric`, `label`, `description`, `match`, `action`, `finally`, there is an increased possibility of the user making a mistake, but this is mitigated by error messages from the LQML parser.

Abstraction measures high level concepts which are not easily grasped by the users, since they do not refer to concrete instances. This dimension thus measures the language's abstraction level.

Assessment: In LQML, we try to keep the number of keywords at the pattern matching feature to a minimum, such that users can have full control on their declarative patterns. In this way there is a very low level of abstraction.

Secondary Notation indicates the availability of options for encoding extra context information within the syntax itself, such as comments.

Assessment: A definition requires a description; further important information can be added in an unstructured way as comments (starting with `#`, extending to the end of the line).

Closeness of Mapping measures the degree of similarity between the representation language and the real-world domain.

Assessment: Our aim is to try to simplify the definition of metrics as much as possible, keeping in mind that possible non-Java experts are using this tool. Despite having this beneficial feature that widens the tool's audience, expert users who require to create more complex metrics, for example, calculating the response time of a server serving a resource, must implement LQML extension functions in Java; metrics with complex matching conditions and actions will even have to be implemented completely in Java.

Consistency measures the usability of the language; in other words, how easy is it for a user to write similar blueprints once the notation pattern has been learned.

Assessment: Unlike in the error-proneness dimension, we here consider that the fixed syntax structure is actually a feature, in a way that consistency is kept for all blueprint definitions.

Diffuseness measures the space required by the notation; i.e. the amount of work-space occupied by the language.

Assessment: Although the blueprints themselves have a clear goal, the rules within the definition might be messy and unclear since different conditions have to be defined in brackets. In LQML, users have to define the precedence of evaluating the conditions (using brackets). The fact that LQML blueprints are defined in a simple text editor means that users might find some difficulty in understanding a rule.

Progressive Evaluation measures the understandability of the language even for a solution that is incomplete. The possibility to try out a partial solution helps users in further understanding their work

Assessment: It is possible to incrementally refine definitions by, e.g., starting with a partial match and a simple 'count' action, and then to further refine the matching pattern by adding conditions, and to define a more complex action.

Role Expressiveness indicates the language's notation and its expressiveness vis-à-vis the whole solution.

Assessment: Our tool is aimed towards the definition of quality metrics for linked data.

In a definition, all required information is adequately labelled to enable easy identification.

Visibility measures the degree of visibility of the language’s notation. If concepts are encapsulated into concepts of a more abstract level, this reduces the visibility of the notation.

Assessment: All available notation is directly visible to the user.

Provisionality measures the ability of the language to allow users to explore potential options.

Assessment: Similarly to the secondary notation dimension, potential options can be explored by temporarily commenting out parts of a definition.

Together the assessment w.r.t. these dimensions provides a comprehensive heuristic guide of LQML, particularly focusing on language features that have not been implemented in an immediate response to the given quality assessment requirements. From this evaluation we can identify certain problems in the current implementation of the syntax, such as the possibility of reusing components of blueprints within others. These heuristics also stress the importance of the need of a better presentation view tool (graphical interface) for the user, while also highlighting that whilst we are widening the scope of metric definition for non-Java experts, we are limiting ourselves to simple pattern matching metrics and thus more complex metrics cannot be defined. These measurements will help us in the second phase of the language definition.

5 State of the Art

A Domain Specific Language (DSL) is a small declarative programming language focusing on a particular domain, offering appropriate notations and abstractions, in a way that is easy to use for non-programmers [8,13,18]. The authors of [8,13] describe a number of benefits of DSLs, including:

- the enhancement of productivity;
- the incorporation of domain knowledge;
- the possibility of portability;
- the understandability of declarative programs by domain experts themselves;
- easily maintainable code.

A DSL development methodology starts with the *decision* stage, where stakeholders decide if the effort in investing in a new DSL pays off in the future. If the stakeholders decide to go ahead, then they proceed to the *analysis* stage, where the problem domain is identified and knowledge about that domain is gathered. Following that, the DSL is then *designed* where the knowledge is concisely described as semantic notations and graphically by using tools such as a feature model¹¹. Finally, the DSL is *implemented*. In the article “When and How to Develop Domain-Specific Languages”, Mernik et al. [18] provide the reader with a comprehensive insight on DSL development methodologies, by identifying patterns for the four stages of the development methodology.

¹¹ http://en.wikipedia.org/wiki/Feature_model

In this section we mention a few examples from a growing list of domain specific languages. Each DSL builds on a data model to encapsulate domain knowledge into an abstract notation.

Domain specific languages are popular within various applications. \LaTeX is a document preparation typesetting system usually used for technical and scientific publications. HTML and XML are generic markup languages that are also DSLs. The former is used to generate websites, whilst the latter is used as an interoperable data model. XPath¹² is an expression language enabling the processing of values in an XML data model. XPath uses path expressions to navigate through XML. RuleML¹³ is an XML markup language that allows rules to be defined using a formal notation.

In the Semantic Web there are a number of domain specific languages. The RDF data model¹⁴ is based on triple statements (subject–predicate–object) that enable the description of real-world objects as machine-readable semantic resources. On top of this data model, RDF Schema¹⁵ is a vocabulary that provides a number of classes and properties to describe a resource in RDF. This schema language also provides the basic concepts for the development of new domain specific ontologies. The RDF data model can be serialised in different formats, such as RDF/XML and Turtle. SPARQL¹⁶ is a domain specific query language for querying the RDF data model. The Web Ontology Language (OWL)¹⁷ adds more semantics on top of the RDF Schema. OWL enables users to create inferencing rules and statements on RDF. Going further away from the data model, the Rule Interchange Format (RIF) [15] is a web standard defining an interchange language for rules within different systems to achieve interoperability. It focuses on the definition of various dialects, which enables the exchange of rules within different systems. The above mentioned DSLs are just a few examples tackling different aspects of the RDF data model. Similar to these, the proposed Luzzu Quality Metric Language is also based on this semantic data model.

6 Concluding Remarks

Data quality assessment is crucial for the wider deployment and use of Linked Data. With the Luzzu Quality Metric Language we empower domain experts who are not proficient in using third generation programming languages to define domain specific quality metrics for linked open datasets. We defined the Luzzu Blueprint Ontology to ensure that quality metrics defined with our proposed domain specific language can be shared, queried and reused easily in a semantic manner. LQML was evaluated systematically against the Cognitive Dimensions of Notation, a methodology developed purposely to assess formal notations such as those of programming languages. The evaluation pointed out shortcomings in the current implementation of the DSL and possible future improvements.

¹² <http://www.w3.org/TR/xpath-30/>

¹³ <http://ruleml.org>

¹⁴ <http://www.w3.org/TR/rdf11-primer/>

¹⁵ <http://www.w3.org/TR/rdf-schema/>

¹⁶ <http://www.w3.org/TR/rdf-sparql-query/>

¹⁷ <http://www.w3.org/TR/owl2-overview/>

Together with the Luzzu framework, we see this as the first step to break the poor quality reputation barrier of Linked Open Datasets. Regarding future work, we aim to create an interactive user interface for the definition of LQML metrics in order to visualise and author metrics, and to offer an online pool of quality metrics that can be queried and downloaded into Luzzu, and to which the Linked Open Data Community can contribute. We also plan to refine LQML with more generic keywords that can be used within the `match`, `action`, and `finally` parts of a definition.

References

1. Auer, S., Bühmann, L., Dirschl, C., Erling, O., Hausenblas, M., Isele, R., Lehmann, J., Martin, M., Mendes, P.N., van Nuffelen, B., Stadler, C., Tramp, S., Williams, H.: Managing the life-cycle of linked data with the LOD2 stack. In: Proceedings of International Semantic Web Conference (ISWC 2012) (2012)
2. Bizer, C.: Quality-Driven Information Filtering in the Context of Web-Based Information Systems. Ph.D. thesis, FU Berlin (Mar 2007), http://www.diss.fu-berlin.de/diss/receive/FUDISS_thesis_000000002736
3. Blackwell, A., Britton, C., Cox, A., Green, T., Gurr, C., Kadoda, G., Kutar, M., Loomes, M., Nehaniv, C., Petre, M., Roast, C., Roe, C., Wong, A., Young, R.: Cognitive dimensions of notations: Design tools for cognitive technology. In: Beynon, M., Nehaniv, C., Dautenhahn, K. (eds.) *Cognitive Technology: Instruments of Mind*, Lecture Notes in Computer Science, vol. 2117, pp. 325–341. Springer Berlin Heidelberg (2001), http://dx.doi.org/10.1007/3-540-44617-6_31
4. Blackwell, A., Green, T.: Notational systems – the cognitive dimensions of notations framework (2002)
5. Cyganiak, R., Jentzsch, A.: About the Linking Open Data dataset cloud, <http://lod-cloud.net>
6. Debattista, J., Lange, C., Auer, S.: Representing dataset quality metadata using multi-dimensional views. In: Filipowska, A., Sack, H., Lehmann, J. (eds.) *SEMANTiCS*. pp. 92–99 (2014)
7. Debattista, J., Scerri, S., Rivera, I., Handschuh, S.: Processing ubiquitous personal event streams to provide user-controlled support. In: Lin, X., Manolopoulos, Y., Srivastava, D., Huang, G. (eds.) *Web Information Systems Engineering – WISE 2013*, Lecture Notes in Computer Science, vol. 8181, pp. 375–384. Springer Berlin Heidelberg (2013), http://dx.doi.org/10.1007/978-3-642-41154-0_28
8. van Deursen, A., Klint, P., Visser, J.: Domain-specific languages: An annotated bibliography. *SIGPLAN Not.* 35(6), 26–36 (Jun 2000), <http://doi.acm.org/10.1145/352029.352035>
9. Dürst, M., Suignard, M.: Internationalized resource identifiers (IRIs). RFC 3987, Internet Engineering Task Force (IETF) (2005), <http://www.ietf.org/rfc/rfc3987.txt>
10. Flemming, A.: Quality characteristics of linked data publishing datasources. Master’s thesis, Humboldt-Universität zu Berlin, Institut für Informatik (2011)
11. Hitzler, P., Janowicz, K.: Linked data, big data, and the 4th paradigm. *Semantic Web* 4(3), 233–235 (2013)
12. Hogan, A., Umbrich, J., Harth, A., Cyganiak, R., Polleres, A., Decker, S.: An empirical survey of linked data conformance. *Web Semantics: Science, Services and Agents on the World Wide Web* 14(0), 14–44 (2012), <http://www.sciencedirect.com/science/article/pii/S1570826812000352>

13. Hudak, P.: Domain specific languages. In: Handbook of Programming Languages, Vol. III: Little Languages and Tools, chap. 3, pp. 39–60. MacMillan, Indianapolis (1998)
14. Juran, J.M.: Juran's Quality Control Handbook. McGraw-Hill, 4 edn. (1974), <http://www.amazon.com/exec/obidos/redirect?tag=citeulike07-20&path=ASIN/0070331766>
15. Kifer, M.: Rule interchange format: The framework. In: Calvanese, D., Lausen, G. (eds.) RR. Lecture Notes in Computer Science, vol. 5341, pp. 1–11. Springer (2008), <http://dblp.uni-trier.de/db/conf/rr/rr2008.html#Kifer08>
16. Kontokostas, D., Westphal, P., Auer, S., Hellmann, S., Lehmann, J., Cornelissen, R., Zaveri, A.J.: Test-driven evaluation of linked data quality. In: Proceedings of the 23rd international conference on World Wide Web (2014), http://svn.aksw.org/papers/2014/WWW_Databugger/public.pdf, to appear
17. Le-Phuoc, D., Polleres, A., Hauswirth, M., Tummarello, G., Morbidoni, C.: Rapid prototyping of semantic mash-ups through semantic web pipes. In: Quemada, J., León, G., Maarek, Y.S., Nejdl, W. (eds.) Proceedings of the 17th WWW conference. pp. 581–590. ACM Press (2009)
18. Mernik, M., Heering, J., Sloane, A.M.: When and how to develop domain-specific languages. ACM Comput. Surv. 37(4), 316–344 (Dec 2005), <http://doi.acm.org/10.1145/1118890.1118892>
19. Schmachtenberg, M., Bizer, C., Paulheim, H.: Adoption of the linked data best practices in different topical domains. In: Mika, P., Tudorache, T., Bernstein, A., Welty, C., Knoblock, C., Vrandečić, D., Groth, P., Noy, N., Janowicz, K., Goble, C. (eds.) The Semantic Web – ISWC 2014, Lecture Notes in Computer Science, vol. 8796, pp. 245–260. Springer International Publishing (2014), http://dx.doi.org/10.1007/978-3-319-11964-9_16
20. Zaveri, A., Rula, A., Maurino, A., Pietrobon, R., Lehmann, J., Auer, S.: Quality assessment methodologies for linked open data. Semantic Web Journal (2014), <http://www.semantic-web-journal.net/content/quality-assessment-linked-data-survey>